REMARKS

Applicants note the title of the application was changed in an amendment filed on November 26, 2003 but that subsequent amendments and related filing papers may have incorrectly used the old title.

Claims 1-5, 13-16, 19 and 20 are pending but were rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 6,018,617 (*Sweitzer*) and U.S. Patent No. 5,657,256 (*Swanson*). Applicants request entry of the amendments above and withdrawal of the rejections. Claims 1, 3-4, 14, and 16 are amended in this response merely for brevity.

Independent claims 1, 5, and 14

Applicants submit that independent claims 1, 5, and 14 are not rendered unpatentable by the cited prior art because these claims include elements neither taught nor suggested by the cited prior art. More particularly, Applicants submit that *Sweitzer* and *Swanson* fail to disclose or fairly suggest, among other things, "generating a test item variant of the test item by assigning values to the variables using a simultaneous constraint solver" as recited in claim 1. Similarly, the cited prior art fails to teach or suggest "using a simultaneous constraint solver to determine values for the variables based on the constraints" as recited in claim 5, and "simultaneously solving test item model constraints pertaining to variables of the selected test item model" as recited in claim 13.

Sweitzer is directed to a method and a system for producing tests that includes the capability to format mathematical expressions. An authoring tool uses variation rules, which include an ordered list of definitions and constraints, to define instances of generalized problems. "To produce an instance of a problem, the list of variation rules is evaluated sequentially from top to bottom. If a constraint is not satisfied, the current pass through the list is abandoned and evaluation restarts from the top of the list. A valid instance of the problem results when the end of the variation rule is reached." *Sweitzer* at 12:41-46. In other words, *Sweitzer* uses a **sequential** constraint solver that "processes the variation rules for a problem from the top down." *Id.* at 17:49-50.

Applicants have previously amended claim 1 to clarify that more than one constraint is evaluated simultaneously. Thus, in contrast to the sequential constraint solver of *Sweitzer*, claim 1 requires the use of a **simultaneous** constraint solver to resolve a plurality of constraints in order to generate test item variants. A simultaneous constraint solver solves for all constraints simultaneously. In other words, a simultaneous constraint solver merely requires determining

values for a set of constraints once to generate a test item variant. As a result, test items are generated more efficiently. In contrast, the sequential constraint solver of *Sweitzer* deletes computed constraint values and restarts from the beginning of its constraint sequence when a constraint is not satisfied. *Sweitzer* at 15:39-40. As such, *Sweitzer*, unlike the simultaneous constraint solver of claims 1, 5, and 13, typically requires evaluating multiple constraints a plurality of times until all of the constraints in the sequence are satisfied. Accordingly, *Sweitzer* does not disclose "generating a test item variant of the test item by assigning values to the variables using a simultaneous constraint solver" as required by claim 1. Claims 5 and 13 are similarly distinct.

The Examiner asserts that *Swanson* cures the shortcomings of *Sweitzer* by disclosing a test item system that solves for multiple constraints (column 8 lines 5-47). *Swanson* discloses only that an important quality of a test construction model is "coming close as possible to meeting all constraints simultaneously" (column 8 lines 5-15). While this may be an admirable goal, Applicants respectfully disagree with any assertion that the combination of *Sweitzer* and *Swanson* provide any implemented solution that includes a simultaneous constraint solver as provided by the present invention.

Indeed, *Swanson* actually teaches away from the assertion that it would be obvious to one of ordinary skill to modify *Sweitzer* to "use a constraint solver to satisfy or simultaneously solve for multiple constraints". See for example column 6 line 65 to column 7 line 5 of *Sweitzer*, which are statements that note the infeasibility of a prior art approach to a solution to the recognized problem, as the Examiner notes in the most recent office action. However, neither *Swietzer* nor *Swanson* then proceed beyond the decried prior art limitations regarding simultaneous constraint solution, except to note that the problem is "difficult or impossible to satisfy" with binary programming models (*Swanson* column 8 lines 5-10 and column 6 line 65 to column 1 line 1, as cited), and that a less than complete solution that softens constraints to mere "desired properties" (*Swanson* column 8 line 12) may be the best they can achieve.

Dependent claims 2-4, 14-17, and 19-20

Claims 2-4 (which depend on claim 1) and claims 14-17 and 19-20 (which depend on claim 13) are patentable for at least the same reasons as their parent claims. Claim 17 warrants particular discussion. Applicants respectfully disagree with the assertion that it would have been

"an obvious design choice to a person of ordinary skill in the art to use C++ and variation rules language to simultaneously solve test item model constraints" and that using PROLOG IV and Test Creation Assistant (TCA) constraint language is not for a particular purpose nor solving a stated problem.

The specification provides that the present invention's preferred embodiments use PROLOG as its simultaneous constraint solver (page 41 lines 1-2). The specification states on page 45 lines 5-10 that the "TCA constraint solver can solve linear constraints and a large class of nonlinear constraints" and "returns all the solutions to the specified constraints", i.e. the invention is for a particular purpose and is solving the stated problem. On page 62 line 22, the specification clearly notes that "Constraints are solved all at once as a whole." This point is made yet again from page 66 line 19 to page 67 line 4, quoted here:

> *"8. Constraints are Solved as a Whole.*
>
> *All the constraints specified for one constraint are all solved as a whole, and not partially. This is particularly important in the case of the TCA where constraints are entered on different lines without any explicit operators (e.g. comma or semicolon) combining them (TCA supplies the default comma-operator (i.e. conjunct) between adjacent constraints) and thus one might get the incorrect impression that the constraints are solved independently."*

In particular, page 41 lines 15-18 of the specification denote that a specified expression scanner is generated that "breaks the mathematical constraints into individual words and operators (called tokens) which are further used by the Prolog-expression parser." Page 42 lines 2-8 describe that the parser "recognizes the syntactic patterns of mathematical constraints" and "transforms the mathematical constraints into appropriate Prolog clauses, and calls Prolog IV to solve those clauses." An API is also provided to allow other programs to "access the constraints-solver to solve mathematical constraints." (page 42 lines 10-14). There is no suggestion of these processes in the prior art.

Contrasting the present invention with conventional programming, on page 43 line 16 to page 44 line 1, the specification details that the "TCA constraint language differs from procedural languages (e.g. C, Fortran) principally in that it is a goal-oriented language, that is users need specify only the constraints to be solved, and not how to solve them." C++ is a procedural language. Further, in TCA, program flow "is controlled by the constraint solver" and is "constraint-order independent". In contrast, page 63 lines 1-6 explain that unlike TCA, procedural languages have to provide a large number of program-flow control mechanisms. In

lines 8-11 of page 63, the specification explains that because "the TCA constraint language uses relations to implement most functions and operators, one can use the same function [operator] to map a set of arguments to a result, or to map a result back to a set of arguments. ... In procedural languages, one has to explicitly apply the reverse function to achieve the effect illustrated above." Also, as noted in lines 16-17, "TCA constraint language has no value-storage and no assignment." Given these fundamental differences, writing and solving constraints in TCA is not an variation of prior art attempts that would be obvious to one of ordinary skill in the art.

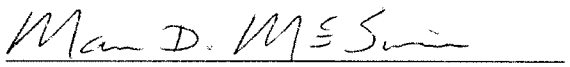
CONCLUSION

In conclusion, Applicants respectfully submit that the application and the claims are in condition for allowance and respectfully request favorable consideration and the timely allowance of all pending claims. By the above amendments, Applicants submit that no new matter has been added to the application. If, for any reason, the application and claims are not in condition for allowance, or any additional information is required, the Examiner is invited to contact the undersigned at (650) 331-2048.

AUTHORIZATION

The Commissioner is hereby authorized to charge any additional fees which may be required for this Response, or credit any overpayment, to deposit account no. 05-0426.

Respectfully submitted,



Marc D. McSwain
Reg. No. 44,929

Date: August 25, 2008

CUSTOMER NO. 26565
MAYER BROWN LLP
P.O. Box 2828
Chicago, Illinois 60690-2828
(650) 331-2000